Chapter 6

Graphics

R is a fantastic tool when it comes to graphics. An entire short course could be devoted to making graphics and plotting in R, as there are many approaches and packages designed for this specific task. Unfortunately, we can only scratch the surface and the main purpose of this chapter is to showcase some of R's graphical capabilities and to point you in the right direction.

R comes with a slew of graphic capabilities and functions pre-installed as part of the `base` package. Many of these functions are well documented and I encourage you to learn about them and play around with various approaches by browsing the following website: http://gallery.r-enthusiasts.com/.

I believe that ultimately the choice of approach comes down to taste. If you are into a simple look, that may or may not be terrible consistent across different types of plots (e.g. histograms vs. pie charts) than the `base` graphics functions may work for you. At the end of the day any approach you may want to take requires some learning and practice. With this said, you might as well begin with a more flexible and consistent approach – `ggplot2`.[1] So let's use Hadley Wickham's `ggplot2` package. Extensive and detailed documentation and examples for his package can be found here: http://docs.ggplot2.org/current/.

6.1 ggplot2

For all examples we will use the fuel economy data found on my website. Let's load it up and also install and load the `ggplot2` package.

```
1 > FE2013 <- read.csv("http://peterhaschke.com/Teaching/R-
    Course/FE2013.csv")
2 > install.packages("ggplot2") # this may be installed in the
    starlab already
3 > library(ggplot2)
  >
```

[1] The learning curves for plotting with the `base` functions vs. `ggplot2` or `lattice` are comparable. You should start learning the approach that you find most appealing, visually.

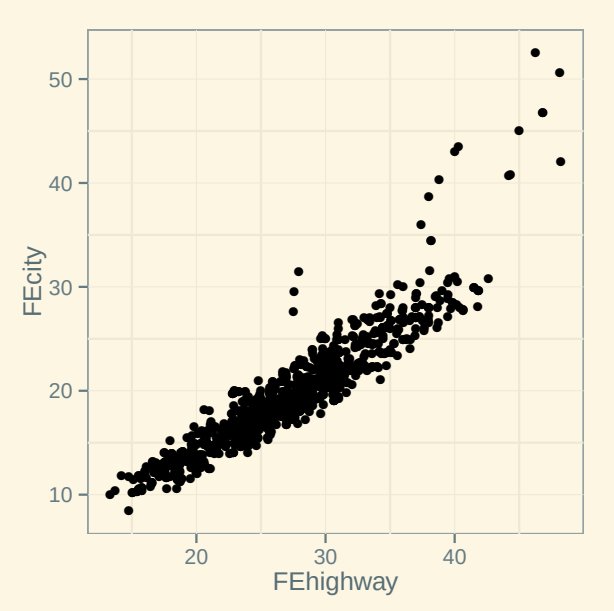
6.1.1 Scatterplots

Having installed and loaded ggplot2 we now have access to the packages `ggplot()` function. The `ggplot()` function does nothing other than create or initiate a ggplot object. It takes at the minimum one argument: `data`. All aspects of a plot are then subsequently added as layers with separate functions called `geoms`. The geom used to create scatterplots is `geom_point()`. The `geom_point()` function itself takes an argument called `aes` which assigns data to aesthetic properties. This sounds terrible complicated but really isn't. Let's give it a shot.

```
> plot1 <- ggplot(data = FE2013) # we are creating an empty
   ggplot object
> plot1 # if you print this plot to the screen, R's graphic
   device will open an empty plot
>
```

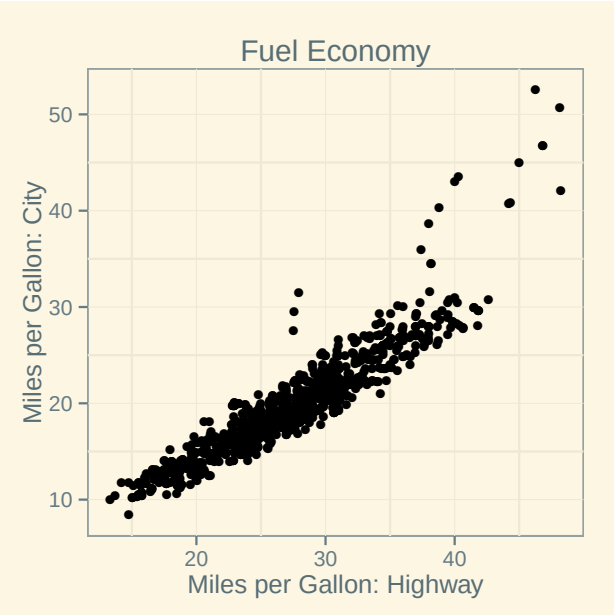
We now literally add a scatterplot layer to `plot1` via the `geom_point()` function. The `geom_point()` function takes the `aes()` argument, which generates a mapping describing how variables in the data are mapped to visual properties of `geoms`. For `geom_point()` the arguments `x` and `y` must be supplied to `aes`. Let's compare highway and city fuel economies.

```
> plot1 <- plot1 + geom_point(aes(x = FEhighway, y = FEcity))
> plot1 # we now have added a layer and printing the object
   to the screen will open R's graphics device with a
   scatterplot.
>
```



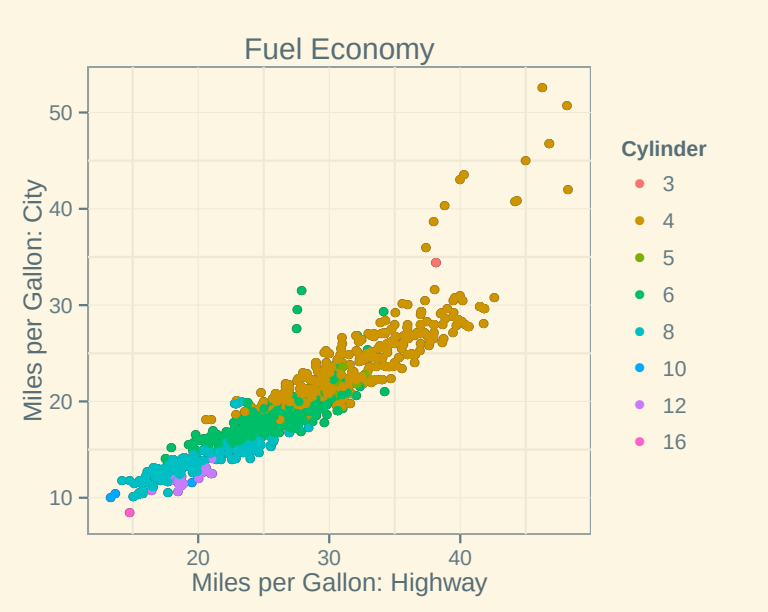
The plot is still pretty bare and poorly labeled. We can change all this by adding more layers. Let's add a layer called `labs()`. It takes the arguments `x`, `y`, and `title`.

```
> plot1 <- plot1 + labs(title = "Fuel Economy", x = "Miles
   per Gallon: Highway", y = "Miles per Gallon: City") #
   saving the labs layer to plot1
> plot1
>
```



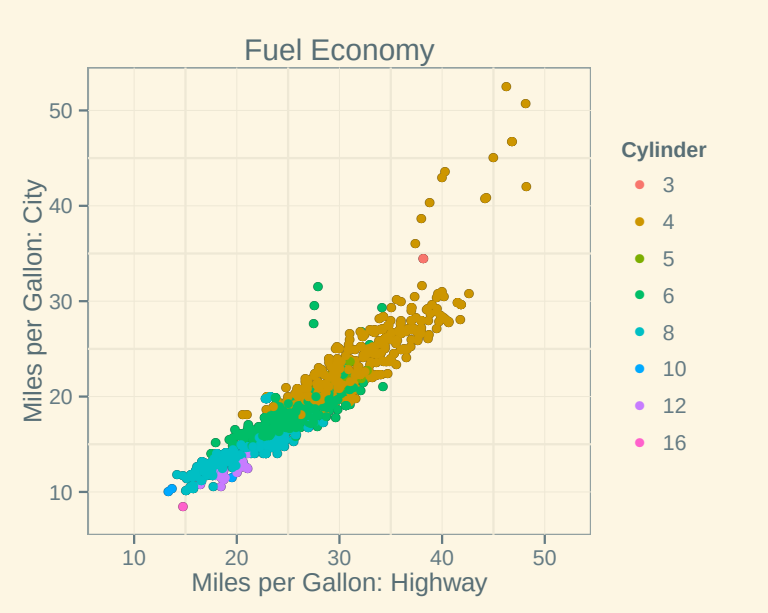
The great thing about ggplot2 is that you can map additional data to aesthetics easily. Let's add a factor to the mapping via the `color` aesthetic. This will change the color of the points depending on a factor vector. In our dataset we have a variable called `Cylinder` that seems to be categorical but is currently an integer vector.

```
> class(FE2013$Cylinder)
[1] "integer"
> FE2013$Cylinder <- as.factor(FE2013$Cylinder) # let's
   change the Cylinder variable to a factor. note: we are
   overwriting the original variable
> plot1 <- plot1 + geom_point(aes(x = FEhighway, y = FEcity,
   color = Cylinder))
> plot1
```



Very nice. The plot looks pretty good. But just for good measure let's change the axis tick-marks. To change the scales layer we can use the `scale_x_continuous()` and `scale_y_continuous()` functions. Each takes an argument `breaks` which can be supplied with a vector of tick-mark locations. To change the limits of the axes we can use the `coord_cartesian()` function which take the argument `xlim` and `ylim`.

```
> plot1 <- plot1 + coord_cartesian(xlim = c(5,55), ylim = c
  (5,55)) # this changes the limits of the axes
> plot1 <- plot1 + scale_x_continuous(breaks = c(10, 20, 30,
  40, 50)) # this changes the tick-marks on the x-axis
> plot1
>
```



6.1.2 Exporting Graphics

To export plots you have created in R to a format that you can use in your LaTeX documents you should use the `pdf()` function to save the plot as a `.pdf`.[2] The `pdf()` function will initiate the R graphics device and then save your plot to disk. There is a sequence that you need to follow in your code. First, you need to initiate the graphics device via the `pdf()` function. Secondly, you print the plot and all layers you have created to the device so that the `pdf()` function will save it. Lastly, you will have to close the graphics device with the `dev.off()` function. The `pdf()` function takes a variety of arguments. The most useful ones are: `file` to specify the location where your file will be saved, and `width` and `height` to set the size of the `.pdf` file. Let's save the plot we have just created.

```
> pdf(file = "Z:/Plot1.pdf", width=5, height=4) # Step 1
> plot1 # Step 2
> dev.off() # Step 3
null device
          1
>
```

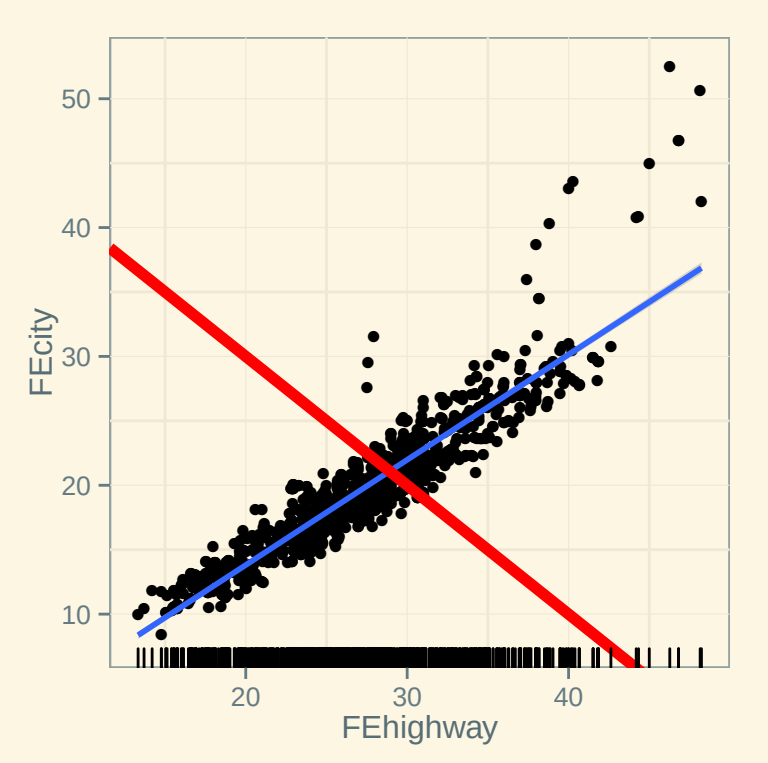
6.1.3 Adding more geoms

For exposition, the code for our figure so far has been unnecessarily complicated. Instead of creating a `ggplot` object and then successively saving additions to our plot, we can simplify things and add them all at once (i.e. we can write: `Plot <- ggplot(...) + geom(...) + labs()`). As long as each line we feed to R ends with a "+" R will know more input is coming. Let's go back to our basic plot and add geoms in one swoop. We will add a best fit line to the figure (via

[2] For exporting to other filetypes you can you the `jpeg()`, `ps()`, or `png()` functions.

the geom_smooth() function), an arbitrary line (via the geom_abline() function) and a rug plot (via the geom_rug() function). Note: that we are specifying the the aes() argument which maps the data to an aesthetic object, directly in the ggplot() function instead of separately in each geom.

```
> plot2 <- ggplot(data = FE2013, aes(x = FEhighway, y =
   FEcity)) +
      geom_point() +
      geom_smooth(method = "lm", size = 1) +
      geom_abline(intercept = 50, slope = -1, color = "red",
         size = 2) +
      geom_rug(sides = "b")
> plot2
>
```



6.1.4 Boxplots: `geom_boxplot()`

The ggplot2 package is quite powerful but as you can see it is also somewhat complicated to figure it all out. The remaining sections will provide some templates and examples of what ggplot2 can do for you.

```
> plot3 <- ggplot(data = FE2013, aes(x = Cylinder, y = FEcity
    )) +
      geom_boxplot() +
      labs(y = "Miles per Gallon: City", x = "# Number of
         Cylinders")
> plot3
```